بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

Unit test کردن controller ها در ASP.NET Web API 2

مدرس : مهندس افشین رفوآ

دوره آموزشی Web API

این مقاله بعضی تکنیک های خاص unit test کردن controller ها در Web API 2 را توضیح می دهد. البته پیشنهاد می کنیم قبل از مطالعه این بخش ، مقاله قسمت قبلی را مطالعه کنید.

الگوی معمول در unit test ها، arrange-act-assert می باشد:

- Arrange: هر پیش نیازی را برای اجرا کردن test تنظیم می کند.
- Act: test را اجرا می کند.
- Assert: موفقیت آمیز بودن test را تائید می کند.

مرحله arrange تعداد وابستگی ها (dependencies) را کم می کند و test کردن روی یک چیز صورت می گیرد.

در اینجا مقادیری که باید در controller برنامه Web API شما unit test شود می بینید:

- Action مقدار صحیح پاسخ را بر گرداند.
- پارامتر های نامعتبر پاسخ مناسب خطا را بر گردانند.
- Action متد صحیح در repository یا لایه سرویس را فراخوانی کند.
- اگر پاسخ شامل یک domain model باشد، نوع model را بررسی کند.

این ها مواردی می باشد که باید test شود اما جزئیات آن به پیاده سازی controller شما بستگی دارد. به طور خاص، تفاوت زیادی بین برگرداندن HttpResponseMessage یا IHttpActionResult در action کنترولر وجود دارد.

Test کردن action هایی که HttpResponseMessage بر می گردانند

یک مثال از controller که action آن HttpResponseMessage بر می گرداند در زیر نشان داده شده است:

```
public class ProductsController : ApiController
{
    IProductRepository _repository;
    public ProductsController(IProductRepository repository)
    {
        _repository = repository;
    }
    public HttpResponseMessage Get(int id)
    {
        Product product = _repository.GetById(id);
        if (product == null)
        {
            return Request.CreateResponse(HttpStatusCode.NotFound);
        }
        return Request.CreateResponse(product);
    }
    public HttpResponseMessage Post(Product product)
    {
        _repository.Add(product);
        var response = Request.CreateResponse(HttpStatusCode.Created, product);
        string uri = Url.Link("DefaultApi", new { id = product.Id });
        response.Headers.Location = new Uri(uri);
        return response;
    }
}
```

توجه داشته باشید که controller از dependency injection برای تزریق(inject) IProductRepository استفاده می کند. این عمل controller را آزمایش پذیر تر می کند(testable) زیرا شما می توانید یک mock repository تزریق کنید. Unit test زیر تائید می کند که متد Get یک Product در پاسخ می نویسد. در اینجا فرض می کنیم repository یک IProductRepository ساختگی (mock) می باشد.

```
[TestMethod]
public void GetReturnsProduct()
{
    // Arrange
    var controller = new ProductsController(repository);
    controller.Request = new HttpRequestMessage();
    controller.Configuration = new HttpConfiguration();
    // Act
    var response = controller.Get(10);
    // Assert
    Product product;
    Assert.IsTrue(response.TryGetContentValue<Product>(out product));
    Assert.AreEqual(10, product.Id);
}
```

قرار دادن Request و Configuration در controller خیلی مهم است. در غیر این صورت، test خطای ArgumentNullException یا InvalidOperationException ارسال می کند.

Test کردن link ساز

متد Post، UrlHelper.Link را برای ساخت لینک ها در پاسخ فراخوانی می کند. این کار به یک سری تنظیمات در unit test نیاز دارد:

کلاس UrlHelper به درخواست URL و داده مسیر نیاز دارد و آنها را مقداردهی می کند. مورد بعدی stub UrlHelper یا ساختگی (mock) می باشد. با این رویکرد، شما می توانید مقادیر پیش فرض ApiController.Url را با یک ورژن mock یا stub که یک مقدار ثابت بر می گرداند عوض کنید.

```
[TestMethod]
public void PostSetsLocationHeader()
{
    // Arrange
    ProductsController controller = new ProductsController(repository);
    controller.Request = new HttpRequestMessage {
        RequestUri = new Uri("http://localhost/api/products")
    };
    controller.Configuration = new HttpConfiguration();
    controller.Configuration.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional });
    controller.RequestContext.RouteData = new HttpRouteData(
        route: new HttpRoute(),
        values: new HttpRouteValueDictionary { { "controller", "products" } });
    // Act
    Product product = new Product() { Id = 42, Name = "Product1" };
    var response = controller.Post(product);
    // Assert
    Assert.AreEqual("http://localhost/api/products/42",
response.Headers.Location.AbsoluteUri);
}
```

حالا می خواهیم test را با فریم ورک Moq بسازیم:

```
[TestMethod]
public void PostSetsLocationHeader_MockVersion()
{
    // This version uses a mock UrlHelper.
    // Arrange
    ProductsController controller = new ProductsController(repository);
    controller.Request = new HttpRequestMessage();
    controller.Configuration = new HttpConfiguration();
    string locationUrl = "http://location/";
```

```
    // Create the mock and set up the Link method, which is used to create the Location
header.
    // The mock version returns a fixed string.
    var mockUrlHelper = new Mock<UrlHelper>();
    mockUrlHelper.Setup(x => x.Link(It.IsAny<string>(),
It.IsAny<object>())).Returns(locationUrl);
    controller.Url = mockUrlHelper.Object;
    // Act
    Product product = new Product() { Id = 42 };
    var response = controller.Post(product);
    // Assert
    Assert.AreEqual(locationUrl, response.Headers.Location.AbsoluteUri);
}
```

در این ورژن، لازم نیست هیچ گونه تنظیمی برای داده های مسیر (route data)اجرا کنید زیرا mock UrlHelper یک رشته ثابت بر می گرداند.

نکته: در این مقاله از Moq استفاده کردیم ولی هر گونه فریم ورک mock را می توانید استفاده کنید.

Test کردن action هایی که IHttpActionResult بر می گردانند

در Web API 2 یک action کنترولر، IHttpActionResult بر می گرداند که شبیه ActionResult در Asp.net MVC می باشد. رابط IHttpActionResult یک الگوی متنی برای ساختن پاسخ های HTTP تعریف می کند. به غیر از ساختن پاسخ به صورت مستقیم، controller یک IHttpActionResult بر می گرداند. سپس، pipeline برای ساختن پاسخ، IHttpActionResult را فراخوانی می کند. این رویکرد باعث ساده تر شدن نوشتن unit test ها می شود زیرا می توانید از بسیاری تنظیمات برای **HttpResponseMessage نیاز است عبور کند.**

در اینجا یک مثال از controller که action آن IHttpActionResult بر می گرداند، نشان داده شده است:

```
public class Products2Controller : ApiController
{
    IProductRepository _repository;
    public Products2Controller(IProductRepository repository)
    {
        _repository = repository;
    }
    public IHttpActionResult Get(int id)
    {
        Product product = _repository.GetById(id);
        if (product == null)
        {
            return NotFound();
        }
        return Ok(product);
    }
```

```
    public IHttpActionResult Post(Product product)
    {
        _repository.Add(product);
        return CreatedAtRoute("DefaultApi", new { id = product.Id }, product);
    }
    public IHttpActionResult Delete(int id)
    {
        _repository.Delete(id);
        return Ok();
    }
    public IHttpActionResult Put(Product product)
    {
        // Do some work (not shown).
        return Content(HttpStatusCode.Accepted, product);
    }
}
```

در این مثال چند الگوی معمول با استفاده از IHttpActionResult نشان داده شده است. حالا می خواهیم نحوه unit test کردن آن را ببینیم.

Action مقدار ۲۰۰(ok) را با یک پاسخ بر گرداند

اگر product پیدا شود، متد Get مقدار ok(product) را فراخوانی می کند. در unit test مطمئن شوید مقدار بازگشتی OkNegotiatedContentResult می باشد و product نمایش داده شده دارای ID صحیح باشد.

```
[TestMethod]
public void GetReturnsProductWithSameId()
{
    // Arrange
    var mockRepository = new Mock<IProductRepository>();
    mockRepository.Setup(x => x.GetById(42))
        .Returns(new Product { Id = 42 });
    var controller = new Products2Controller(mockRepository.Object);
    // Act
    IHttpActionResult actionResult = controller.Get(42);
    var contentResult = actionResult as OkNegotiatedContentResult<Product>;
    // Assert
    Assert.IsNotNull(contentResult);
    Assert.IsNotNull(contentResult.Content);
    Assert.AreEqual(42, contentResult.Content.Id);
}
```

توجه داشته باشید unit test در action result اجرا نمی شود. می توانید فرض کنید action result یک پاسخ HTTP مستقیم می سازد.(به همین دلیل است که فریم ورک Web API ، Unit test مربوط به خود را دارد!)

Actionمقدار ۴۰۴ (عدم پیدا شدن) بر گرداند

اگر product پیدا نشده باشد متد Get مقدار NotFound() را فراخوانی می کند. برای این مورد، unit test مقدار بازگشتی را برای وجود NotFoundResult کنترل می کند.

```
[TestMethod]
public void GetReturnsNotFound()
{
    // Arrange
    var mockRepository = new Mock<IProductRepository>();
    var controller = new Products2Controller(mockRepository.Object);
    // Act
    IHttpActionResult actionResult = controller.Get(10);
    // Assert
    Assert.IsInstanceOfType(actionResult, typeof(NotFoundResult));
}
```

Action مقدار ۲۰۰(OK) را بدون پیام پاسخ خاصی بر گرداند

متد Delete برای برگرداندن یک پاسخ خالی HTTP 200، ok() را فراخوانی می کند. همانند مثال قبل، unit test ، مقدار بازگشتی را برای مقدار OkResult کنترل می کند.

```
[TestMethod]
public void DeleteReturnsOk()
{
    // Arrange
    var mockRepository = new Mock<IProductRepository>();
    var controller = new Products2Controller(mockRepository.Object);
    // Act
    IHttpActionResult actionResult = controller.Delete(10);
    // Assert
    Assert.IsInstanceOfType(actionResult, typeof(OkResult));
}
```

Action مقدار ۲۰۱(ساخته شده) را با یک header مکان(loacation) بر گرداند

متد Post برای برگرداندن پاسخ HTTP 201 با یک URI در header مکان(location) مقدار CreatedAtRoute را فراخوانی می کند. در unit test، بررسی می کند action مقادیر صحیح مسیریابی (Routing) را اضافه کرده است یا خیر.

```
[TestMethod]
public void PostMethodSetsLocationHeader()
{
    // Arrange
    var mockRepository = new Mock<IProductRepository>();
    var controller = new Products2Controller(mockRepository.Object);
    // Act
    IHttpActionResult actionResult = controller.Post(new Product { Id = 10, Name = "Product1"
});
    var createdResult = actionResult as CreatedAtRouteNegotiatedContentResult<Product>;
    // Assert
    Assert.IsNotNull(createdResult);
    Assert.AreEqual("DefaultApi", createdResult.RouteName);
```

```
    Assert.AreEqual(10, createdResult.RouteValues["id"]);
}
```

Action یک 2xx دیگر با یک پاسخ بر گرداند

متد Put برای برگرداندن یک HTTP ۲۰۲(پذیرفته شده) Content را فراخوانی می کند. به نظر می رسد در این مورد مقدار ۲۰۰(ok) را بر می گرداند اما unit test باید وضعیت کد (status code) را کنترل کند.

```
[TestMethod]
public void PutReturnsContentResult()
{
    // Arrange
    var mockRepository = new Mock<IProductRepository>();
    var controller = new Products2Controller(mockRepository.Object);
    // Act
    IHttpActionResult actionResult = controller.Put(new Product { Id = 10, Name = "Product"
});
    var contentResult = actionResult as NegotiatedContentResult<Product>;
    // Assert
    Assert.IsNotNull(contentResult);
    Assert.AreEqual(HttpStatusCode.Accepted, contentResult.StatusCode);
    Assert.IsNotNull(contentResult.Content);
    Assert.AreEqual(10, contentResult.Content.Id);
}
```